Scilight

Artificial Intelligence and Emerging Technologies
https://www.sciltp.com/journals/aiet

Article

Deep Learning-Based Intrusion Detection System for Cloud Infrastructure Security

N. Moorthy

Computer Applications, Nandha Arts and Science College, Erode 638052, India

How To Cite: Moorthy, N. Deep Learning-Based Intrusion Detection System for Cloud Infrastructure Security. *Artificial Intelligence and Emerging Technologies* **2026**, *3*(1), 4. https://doi.org/10.53941/aiet.2026.100004

Received: 21 January 2026
Revised: 30 March 2026
Accepted: 30 March 2026
Published: 31 March 2026

Abstract: Cloud infrastructure security has become a main issue that needs to be addressed since organizations are moving their sensitive operations to the cloud and thus are exposing their systems to different kinds of cyber threats like DDoS attacks, brute-force attempts, port scans and insider misuse. Traditional rule-based intrusion detection systems do not adapt to changing attack patterns most of the times; thus, advanced solutions should be developed. One of the key contributions of this research work is a Deep Learning-Based Intrusion Detection System (DL-IDS) utilizing UNSW-NB15 dataset which is designed for providing realistic traffic patterns including both normal and malicious activities. First step in the methodology comprises of applying preprocessing techniques such as one-hot encoding, normalization and feature selection, then followed by feature extraction through an autoencoder which helps in dimensionality reduction and noise elimination. Next, fully connected deep neural network (DNN) is used used for classification, optimized with Adam algorithm also early stopping to secure strong training. The experimental performance shows fantastic results measured in several metrics with 99.12% accuracy, 98.87% precision, 99.54% recall and 99.21% F1-score. Such outcomes point out the model's ability to operate in middle with respect to sensitivity and specificity which is quite significant to detective reliably the malicious traffic and at same time keep the false alarms very low. High recall figure means that it is very effective in reporting true positives while the precision figure is the corroborating factor of the low rate of clerical mistakes made on the benign traffic. The F1-score is yet another proof of the system's balanced performance, thus, it can be deployed in real-time in cloud environments. In sum, the DL-IDS framework proposed offers a dynamic, scalable, and effective intrusion detection method which overcomes the drawbacks of traditional systems and presents a significant improvement in the area of cloud infrastructure security.

Keywords: Cloud Security Infrastructure; detection systems; anomalies; autoencoder; Deep Neural Network (DNN); Adam optimizer

1. Introduction

Cloud computing infrastructure is the center of the modern world of computing, offering scalable storage, processing, and application services, which are the key advantages of this instrument [1]. The greater the migration of critical operations of organizations to the cloud, the bigger the attack surface will be. This is often due to the flexibility and dispersion of cloud systems, where the traditional security measures are often insufficient [2]. DL has been identified as an effective system used to detect more complicated and dynamic threats in vast regions. Therefore, the DL-IDS represents a highly significant element of the cloud security tool that is bound to be resilient [3].

DDoS, port scans, brute force, insider abuse, and malware injection are security threats that are prevalent in cloud platforms [4]. The reason is that such attacks are often financed by a great number of various hackers, thus

making it difficult to use rule-based detection [5]. Moreover, a large amount of traffic, multi-tenancy, API exposure, and constant scaling of resources are not a few factors contributing to the challenge of intrusion detection [6]. Models can be used to find anomalies with good accuracy using DL in order to analyze logs, network flows, and patterns of user behavior. Therefore, the adaptive capability of detecting threats by changing with new patterns of attacks is offered by DL-IDS.

There are still some limits to DL methods of intrusion detection other than advantages [7]. The models require a substantial number of labeled datasets to train, and obtaining these datasets is difficult in a real cloud setting [8]. The difficulties created by the high computation cost and intricacy of the models may cause delays in real-time threat detection. DL models, as well, may fall into adversarial attacks and manipulate the input data [9]. Therefore, cloud security must be efficient and functional, and this aspect demands regular review and revision of the models.

The proposed framework addresses the following weaknesses: it provides an autoencoder-based feature extraction method and a DNN implemented to detect intrusion in clouds. Autoencoders minimize the dimensions as well as noise energy by generating a small latent representation of the network traffic. The consequence of this is increased classification accuracy, and simultaneously, it does not demand much in the way of computation. These improved features are used by the DNN classifier to do accurate discrimination between normal and all attack classes in the UNSW-NB15 dataset. The structure uses the Adam optimizer and early stopping to complete the training process in a gradual and continuous manner and prevent overfitting. The system has strong detection ability and rarely raises false alerts because it integrates anomaly detection and supervised classification. The peculiarity of this study is the fact that it possesses one DL architecture that is efficient, scalable, and robust simultaneously. It is this property that renders the proposed framework highly suitable in respect to the security of cloud infrastructures in real-time.

Objectives

- Build a DL-based IDS custom-made for cloud environments to effectively identify both normal and malicious traffic patterns.
- Utilize UNSW-NB15 dataset, which contains diverse then cloud-relevant attack scenarios, ensuring the model learns from realistic traffic behavior.
- Apply one-hot encoding, normalization, and feature selection to refine raw input data, reduce redundancy, and improve training efficiency.
- Employ autoencoder-based feature extraction to compress high-dimensional data into informative latent vectors, minimizing noise and improving classification accuracy.
- Implement a dense neural network with the Adam optimizer and early stopping as its training regimen in order to reach high detection performance in terms of accuracy, precision, recall and F1-score.
- Design the IDS to evolve with developing attack patterns, providing real-time, scalable, and adaptive intrusion detection suitable for dynamic cloud infrastructures.

2. Literature Survey

The given segment evaluated the study conducted previously, which attempted to promote the security of the IoT using sophisticated artificial intelligence technologies. The previous studies already noted that the fast-growing number of IoT devices rendered them more vulnerable to cyberattacks that consequently resulted in severe interruptions of services in case the breaches went undetected [10]. Previous solutions had employed DL models to identify the malware in the traffic; however, most of the available methods still had issues because they relied on protocol and were difficult to implement. The issue of maintaining the same level of accuracy in both the real and the simulated attack scenarios had also been raised by the conducting researchers. These challenges raised the awareness that the DL-IDS, which has been suggested above, was in high demand to provide an efficient protocol-independent approach to detecting intrusion to protect the IoT networks. Gollapalli et al. (2024) addressed cloud security challenges by proposing a unified framework that integrated AES with E2E encryption to ensure data confidentiality, integrity, and availability. Using AES-CBC and secure control of encryption keys, the model secured data throughout its lifecycle and reduced latency to 175 ms, outperforming HTTPS and standalone AES solutions [11].

The previous section of the paper evaluated some of the past research that concentrated on the concept of cyber-physical systems as intricate systems of the physical, computational, and communication net [12]. The previous works had already indicated that increased connectivity of CPS not only resulted in a more serious level of security risk but also rendered it more difficult to detect attacks originating in the cyber side. Researchers had already used ML methods in the analysis of the weaknesses of CPS and to identify the types of threats, and others

had already attempted DL methods to detect security breaches and tag malicious URLs. However, the solutions that were presently available to the problem were usually faced with issues of scalability, accuracy, and generality. The described problems verified the effectiveness of the suggested DL-based framework that was intended to enable CPS security on different levels of the system.

The segment in question was a summary of the research conducted previously that indicated the growing seriousness of network attacks and the need to have intelligent IDSs. The scenario was identified that the methods of ML and DL were widely applied to identify and categorize the threats with more accuracy. Besides, the papers had also indicated that the evolving evil activities necessitated additional adaptable detection models [13]. The published publications mostly relied on publicly available datasets, and most researchers remarked that it was necessary to have constantly updated information due to the dynamic patterns of attack. These shortcomings highlighted the importance of the suggested hybrid DL-IDS that was planned on the path to obtain more advanced features and, as a result, enhance the process of detection of the network-level threats. Warehouse efficiency has become critical with the swift growth of e-commerce, especially in developing countries. Yallamelli et al. (2024) proposed the Dynamic Mathematical Hybridized Modeling Algorithm (DMHMA) using a tabu search approach to optimize order batching. The method improved order-picking efficiency by 25%, reduced operational costs by 15%, and enhanced economic performance by 20% in B2C warehouses [14].

The proposed section reviewed previous research papers that examined the vulnerabilities of the fog and edge computing environments that were frequently targeted by cybercriminals because of the insufficiency of resources in the nodes. It has been mentioned in previous studies that IDS play a very crucial role in these environments, but they had to grapple with large-dimensional network data that led to poor detection. Previous publications considered other feature selection and optimization algorithms, and their objective was to solve the curse of dimensionality, although most solutions only offered minor benefits [15]. The use of machine and DL models in intrusion detection classification was also of interest to researchers. These challenges identified the need for the suggested ESOML-IDS model, which entails optimized feature selection and the best detection rates.

The newest technological trends in communication technologies, the Internet of Things (IoT), and cloud computing have added a lot of comfort and effectiveness to the contemporary systems. Yet, they also have come with more and more sophisticated security issues that cannot be overlooked. An increasing amount of literature has pointed out that the risks that are linked to these technologies increase in proportion as the scale and adoption of these technologies increase [16]. Research has been done in the past on the insufficiency of traditional security controls in addressing the increase in the demand of advanced security against sophisticated cyber-attacks. Specifically, the traditional intrusion detection system (IDS) and other security infrastructure usually cannot ensure adequate security against emerging and new threats, which puts cloud-based applications and IoT networks at a high risk. The results mentioned above highlight the necessity of more advanced adaptive security mechanisms, which can dynamically react to new vectors of attack.

Deep learning (DL) and feature selection methods have also been discussed as possible resolutions to the problem of improving cloud-based intrusion detection. A number of studies have used DL algorithms to detect anomalous traffic patterns in network and system behavior and have shown better predictive performance than the traditional machine learning methods. In the same way, the feature selection techniques have been used to reduce the dimensionality of data, remove useless attributes, and improve the model. All these attempts notwithstanding, there are significant deficiencies in most of the available methods. In particular, they tend to be unable to find the most topical features that can add value to intrusion detection and to make the best use of the opportunities of DL classifiers. This deficiency may lead to either overtrained or underperforming models, or computationally inefficient models, especially with large-scale datasets that are commonly utilized in clouds.

Many researchers have often used benchmark datasets like KDDCup-99 and NSL-KDD to test the effectiveness of intrusion detection methods. These data sets can be used to test in a standardized environment, and they can be used to compare different methodologies. Nevertheless, it has been observed that these datasets might not be sufficiently representative of the complexity and variety of threats in the real world and thus the research results may not be generalized. Recent datasets, including CSE-CIC-IDS-2018 have sought to fill this gap by giving more realistic traffic patterns and attack scenarios to enable more robust testing of IDS models. Studies based on these datasets have supported the finding that whereas DL and feature selection can enhance intrusion detection, there is still an urgency to have frameworks that incorporate the techniques in a more efficient manner. The FEFS-DLM framework was proposed to solve the two problems of selecting the features and predicting the intrusion in response to these drawbacks. FEFS-DLM will enhance the accuracy, efficiency, and reliability of cloud-based intrusion detection by optimizing the process of selecting relevant features, as well as taking full advantage of deep learning classifiers. The framework is an important move towards filling the gap that

has been mentioned by previous research, and it provides a more profound solution to the growing security risks in modern network and cloud-based settings.

Hybrid robotics automation for real-time navigation and preventing collisions with obstacles was enhanced through the integration of AutoNav, LIDAR-based SLAM, and DenseNet with Leaky ReLU. The system achieved flexible movement planning with high accuracy, demonstrating 95% gradient efficiency, 0.045 m localization error, and 97.5% pathfinding accuracy, highlighting its efficacy in dynamic environments Sitaraman et al. (2024) [17]. The earlier studies had conducted feature reduction, clustering, and usage of DL models to enhance the correctness of detection, but most of the methods had challenges with data with many variables and complicated attack behaviors. These problems made it clear that the new hybrid DL-IDS was required, which would be aimed at enhancing the extraction of features and, by extension, offer a better threat classification in the cloud.

The discussed part of the paper provided a detailed review of the previous studies that involved ML-like approaches to detecting attacks on the cloud and highlighted that more traditional ones frequently fell into overfitting in addition to other limitations such as the excessive number of computations and convoluted designs. Open-source datasets have been utilized in the previous studies, and they contained NSL-KDD, BoT-IoT, KDD Cup 99, and CICIDS 2017 to be used in validation of the intrusion detection methods [18]. Additionally, there had been earlier investigations into the various types of feature extraction and optimization methods, yet, at that point, infrastructural models could not reach the desired degree of precision when different types of attacks were involved. The latter shortcomings introduced the necessity of the GSCS-IHNN framework that was offered as a single solution and comprised of an improved feature selection and a smart neural structure to ensure that the detection of cyber threats in the cloud became more resilient.

The previous literature has been highly concerned with the issues related to the data security and privacy of the Internet of Things devices deployed in smart houses, which are highly vulnerable because of their lack of computational capacity [19]. Such constraints ensure that it is hard to enact stringent security control protocols on the devices themselves, leaving them vulnerable to numerous cyberattacks. To address these limitations, previous literature has suggested machine learning (ML)-based intrusion detection systems (IDSs) that are placed either on the edge or in the cloud environment.

Through such systems, it is possible to process the data of the IoT centrally and therefore employ significantly more advanced threat detection algorithms than can be run on resource-constrained devices. Nonetheless, there is practically no researcher that has tried to deploy IDSs on the actual devices themselves, and this creates an urgent void in both on-device security and quick reaction against threats. The problem of attaining a high detection accuracy as well as working efficiency in the case of limited resource settings remains an incessant problem, as was pointed out by numerous previous studies.

Moreover, prior research has been devoted to the classification of different forms of cyberattacks on IoT systems. Although these attempts were useful in the context of studying attack patterns, they did not have real-time solutions that could be used to turn on immediate actions at the device level. Smart home devices are still vulnerable to compromise, and unless the technology can sense and stop this process in time, it may lead to data theft or a loss of privacy, or even a more extensive network outage. It is based on these drawbacks that the introduction of the suggested two-layer ML-based IDS was proposed to improve threat identification and response capabilities of smart home IoT systems. The system seeks to combine layered detection and balance between computational efficiency and the desire to have accurate and timely detection.

Earlier research further emphasized that the efficiency of real-time intrusion detection systems is closely linked to the number of features used in the detection process. While including a large number of features may improve predictive accuracy to some extent, it also results in increased computational and storage demands, which can be prohibitive in limited-resource IoT settings. To address this trade-off, many studies have proposed dimension reduction techniques, particularly feature selection, as a way to identify the most informative attributes from high-dimensional datasets. Feature selection not only reduces computational overhead but also helps maintain or enhance the predictive performance of IDS models by eliminating redundant or irrelevant features. These insights underscored the importance of developing detection models that are optimized for both efficiency and performance, especially in the context of real-time IoT network monitoring.

Secure and efficient data transmission in IoT-enabled MANETs was addressed by Basani et al. (2024) through a Centralized Infrastructure-Aware Reliable Data Transaction Model. The framework integrated EPCC for security, GWO for cluster head selection, and AODV routing, achieving higher reliability and privacy than existing methods, with a 93% success rate and 92.5% overall accuracy [20].

The issue of the security vulnerability posed by the extensive use of cloud computing has been discussed in parallel studies. Previous research found out that the traditional security systems in cloud environments were largely ineffective in the detection of intrusion that was accurately determined, and they also tended to produce

false alarms that overwhelmed network analysts [21]. Other previous solutions were contained in the use of deep learning (DL) models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) to enhance the effectiveness of cloud IDS. Although they had potential, these approaches often faced the issue of the need to trade off computation efficiency with performance in detection. The feature selection process has been restated multiple times in the literature as a method to decrease the computational complexity and enhance model accuracy. Benchmarking with datasets, like CICIDS2018, has been especially helpful, but large and complex datasets like them further emphasize the importance of effective feature selection mechanisms.

These shortcomings revealed the need for the proposed Cu-LSTMGRU model, which will enhance the detection of threats in clouds using methods of multidimensional feature selection and sophisticated DL approaches. In this way, the model aims to increase the efficiency as well as effectiveness of intrusion detection within the cloud environment to solve the long-standing gaps established by the previous studies [22]. On the whole, the preceding research gives a straightforward explanation as to why more advanced and resource-conscious IDS paradigms can be designed in both IoT and cloud environments.

In the history of previous works, AI and DL-based methods with the aim of automated irregularity detection were experimented upon, but even now, many of the models suffer from the problem of small-scale ability and are not highly efficient to be trained. These issues, therefore, led to the suggested IIDNet framework, which integrates various methods—dimensionality reduction, hyperparameter optimization, feature engineering, and an enhanced CNN design—and is, therefore, capable of offering a higher degree of detecting power and, at the same time, being less costly to compute.

Previous studies have sought to address these challenges through the deployment of machine learning (ML)-based intrusion detection systems (IDSs) designed to distinguish malicious traffic from legitimate network activity. While these ML approaches have shown promise, a significant number of them have faced difficulties in managing real-time IoT data streams and addressing multiclass attack scenarios, where multiple types of attacks occur simultaneously or in rapid succession. These limitations have highlighted the need for IDS models capable of operating efficiently in resource-constrained IoT environments while maintaining high detection accuracy.

The new section took into consideration the past studies on increasing vulnerability of cloud systems with DDoS attacks, which entirely affected the service. The studies conducted by the ML techniques were not that old, and precision was boasted of for the same DDoS detection. It was also mentioned that it was highly cautious about misclassifications, and it was very difficult to learn what features are significant in the cloud traffic, which has very large dimensions. Without hesitation, feature selection and combined learning approaches had been used previously, yet again, but many of them could not present similar performance. Therefore, their issues encouraged the necessity of a new framework of DDoS detection that should be grounded on making feature selection more efficient and high-quality classifiers to enhance the recognition reliability.

The proposed section has examined past research highlighting the increasing cybersecurity challenges faced by IoT-enabled electric vehicle (EV) systems. As the adoption of electric vehicles grows and the number of EV charging points expands, a corresponding surge in the volume of generated data has been observed. This growth in connected devices and data traffic inherently increases the vulnerability of IoT systems to cyberattacks, as more network entry points become available to malicious actors [23].

Building upon these findings, the proposed DT-RFECV-based NIDS was conceived to address the gaps identified in earlier research. By combining decision tree classifiers with recursive feature elimination and cross-validation (RFECV), the framework is designed to select an optimal set of features, thereby improving both computational efficiency and detection accuracy [24]. This approach enables the system to provide timely and precise classification of IoT network traffic, ensuring that malicious activities are effectively detected while minimizing resource consumption. Moreover, the DT-RFECV-based NIDS offers a practical solution for securing IoT-enabled EV charging infrastructures, which are increasingly targeted by cyber threats due to their growing adoption. Overall, the proposed model reflects a synthesis of insights from previous studies, offering an optimized framework that balances the dual objectives of efficiency and reliable intrusion detection in IoT-based EV networks.

Workload forecasting in autonomic database systems was improved by Parthasarathy et al. (2023) through a hybrid approach combining clustering and evolutionary algorithms. The method classified workloads into OLTP, DSS, and mixed types using historical and real-time data. Adaptive clustering, evolutionary optimization, and case-based reasoning enhanced response time, throughput, resource utilization, and workload classification accuracy [25].

The proposed section examined previous studies and articles that addressed the mounting security issues of IoT systems that have been exacerbated by the proliferation of cybercrimes and the possibility of quantum computing. It was already found that ordinary cryptography and typical intrusion tracking frameworks had difficulties in light and minimal-size IoT gadgets since typical frameworks necessitated substantial information throughputs [26].

Problem Statement

The hasty adoption of cloud infrastructure has not only enhanced the space that can be attacked but has also exposed the systems to various kinds of vulnerabilities such as DDoS attacks, brute force attacks, port scans, malware injection, and insider abuse [27]. The conventional rule-based IDSs cannot yet adapt to the dynamic and distributed attacks, and thus the accuracy is still low and the rate of false alarms is still high [28]. The fact that cloud environments are dynamic and are marked by multi-tenancy, a hefty flow of traffic, and constant expansion of resources only exacerbates the effectiveness of intrusion detection [29]. DL may be considered a possible solution since it can be used to identify highly complex patterns in the network flows and user behavior, but the costs of doing so may still be significant (high costs of computation); large labeled datasets are required, as are vulnerabilities to adversarial inputs, which should be addressed first before DL can be rolled out in this area [30]. Therefore, there is a pressing need to possess an IDS based on DL that is adaptive, scalable, and well-structured and has the capacity to provide cloud infrastructures with guaranteed protection on a real-time basis.

3. Proposed Methodology for DL-Based Intrusion Detection for Cloud

The proposed methodology aims at developing a DL-IDS, which is bound to enhance cloud infrastructure security as represented in the proposed work architecture diagram, Figure 1. The process begins with UNSW-NB15 data, consisting of realistic network traffic that contains normal network traffic and malware traffic. The categorical variables are helpful in one-hot encoding, the normalization of the numeric variables is helpful in the processing of features, and feature selection is helpful in the elimination of data that is irrelevant in the preprocessing phase. Compact and informative features are obtained with the help of an autoencoder. These are fed into a DNN, which is fully connected, and classification is obtained. Early stopping is used to train the Adam optimizer, and the performance of the model is evaluated using such metrics as accuracy, precision, recall, and F1-score, among others.

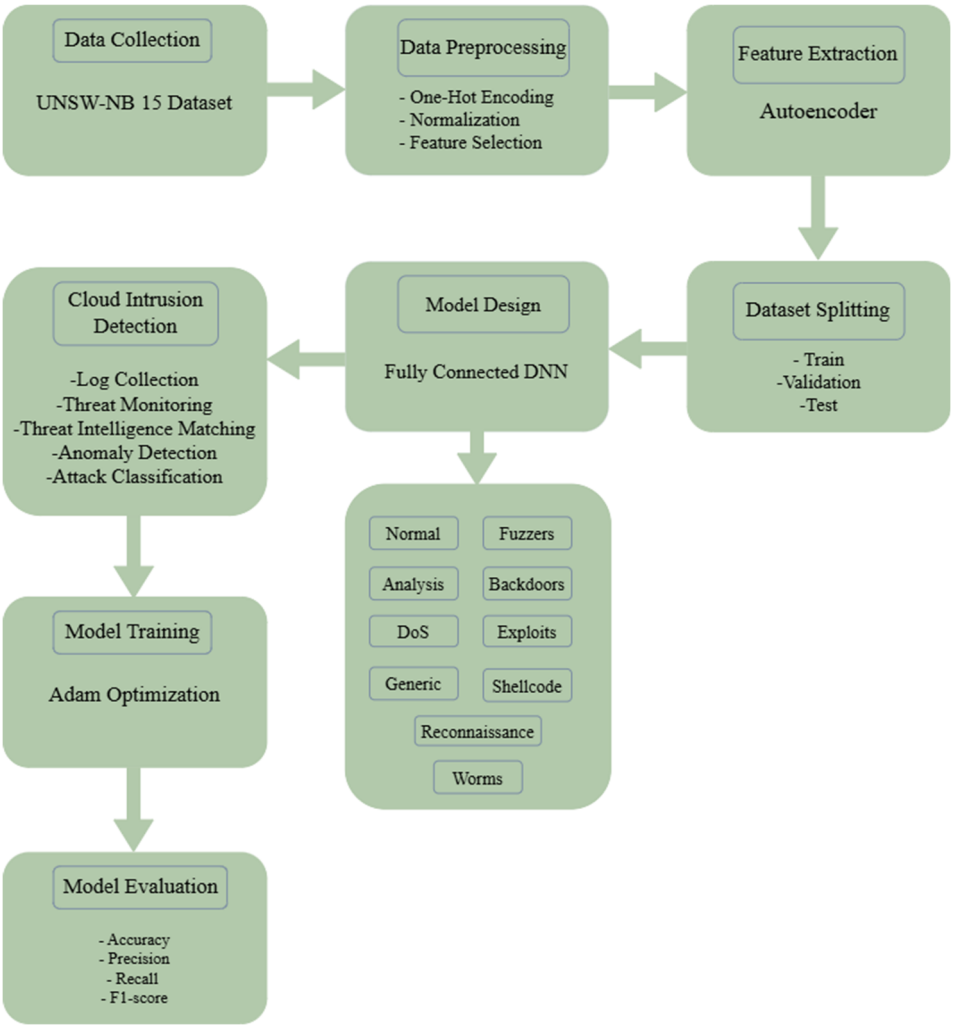

Figure 1. Architecture diagram for planned work.

3.1. Data Collection

The acquisition of cloud-related network traffic of UNSW-NB15 data was the subsequent step in the proposed intrusion detection setup that was preceded by data acquisition. The data is a plentiful blend of normal activities and various attack behaviors generated by the IXIA Perfect Storm tool; therefore, it possesses realistic patterns of traffic in the cloud environment. Every record in the data can be represented by a full configuration of network flow features disclosing protocol behavior, connection properties, and potential malicious motifs. These instances are rich and diverse; hence, the dataset is a good standard to train and test IDSs. The proposed model is appropriate to draw accurate cloud security analysis because it is based on such high-quality and labelled data and is therefore able to differentiate legitimate activities and malicious ones. Dataset link: https://research.unsw.edu.au/projects/unsw-nb15-dataset (accessed on 18 December 2025).

3.2. Data Preprocessing

Preprocessing is the vital process in the entire process of cleaning the database and preparing it to undergo the DL model training, which will be more effective. The first and the simplest is the one-hot encoding, according to which the categorical attributes, such as protocol type, are transformed into binary vectors in such a way that the neural network will not be misunderstood. Subsequently, the continuous variables are normalized so that they are in the same scale and none can have a high value that could prevail, and the learning process can also be stabilized. Adaptive fraud detection was enhanced by Induru and Arulkumaran (2021) through a cybersecurity monitoring framework that integrated GNNs with semantic stream processing. The model combined CEP and time-decayed trust scores over enriched IPv4 logs, reducing false positives by 37% while achieving 94.3% precision and a 92.1% F1 score, with low-latency detection suitable for high-security applications [31].

Finally, Techniques for picking the most important features, like mutual information or step-by-step feature removal can be identified and then applied to data to identify which features are of the most significant significance, and the other features are eliminated since they are considered unnecessary. This entire preprocessing pipeline also results in a rise in the level of training efficiency, a reduction in overfitting, and a major improvement in the performance of the IDS.

3.2.1. One-Hot Encoding

One-hot encoding is a highly important preprocessing technique that is used to change categorical variables into numerical values that can be accepted by DL components [32]. When intrusion detection is detected, the neural networks are not able to process protocol type, service, or state directly since they lack numerical meaning. As a result of this every categorical value is converted into a binary one with one activated element. Mathematically, an m-class categorical variable results in an m-length encoding. This is expressed in Equation (1).

$$\mathrm{OHE}(c_j) = [0,0,\ldots,1,\ldots,0] \tag{1}$$

where, c_j is represented as the selected category to be encoded, m is represented as total number of unique categories in the feature, Index(j) is represented as the position assigned to category c_j in the ordered category list, $\mathrm{OHE}(c_j)$ is characterized as the One-Hot encoded binary vector of length m, v_i is represented as the element at position i in the vector $v_i = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$ and Vector length is represented as equals m, representing the dimensionality of the encoded feature.

3.2.2. Normalization

Normalization is critical in attaining the fact that continuous numerical characteristics have the same impact on the model in training [33]. Since the values of the raw data are normally different, they may cause instability in gradients and, therefore, slow convergence. The solution of min-max scaling is converting the values to a particular range, usually [0,1] [0,1] [0,1], and simultaneously maintaining the relative distribution of the values. Consequently, uniform inputs are provided to models, hence less biased training and improved detection performance. This is shown in Equation (2).

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2}$$

where, x is characterized as original value of feature before scaling, x' is represented as normalized price after min-max scaling, $\min(x)$ is embodied as minimum value of feature within the dataset, $\max(x)$ is represented as

maximum value of feature within dataset, Range is represented as the denominator $\max(x) - \min(x)$, representing the numeric span and Output scale is represented as typically $[0,1]$, but can be adjusted to any desired range.

3.2.3. Feature Selection

The task of feature selection plays a critical role in dimensionality reduction and overall efficacy of the overall system of intrusion detection through the removal of redundant or irrelevant attributes. Such techniques as Mutual Information (MI) and Recursive Feature Elimination (RFE) are still used to extract features that have the most significant contribution to sorting network traffic into normal and harmful activity. This is expressed in Equation (3).

$$S^* = \arg \max_{S \subseteq F} I(S; Y) \tag{3}$$

where, S^* is represented as optimal subset of selected features, F is represented as full set of available features in the dataset, $S \subseteq F$ is represented as candidate subset of features evaluated during selection, Y is represented as target class labels (normal or specific attack types), $I(S; Y)$ is represented as mutual information measuring dependency between features and labels and Optimization Objective is represented as maximizing $I(S; Y)$ ensures only highly relevant features are retained.

3.3. Feature Extraction

Feature extraction using an Autoencoder (AE) has been shown to effectively transform Highly detailed input data into a compressed and meaningful latent representation, as demonstrated by Valivarthi et al. (2024) In this approach, the AE architecture employed an encoder to compress input features into a lower-dimensional vector and a decoder to reconstruct the original data. During training, the model retained only the most significant patterns while minimizing reconstruction error, thereby eliminating noise and redundant information [34]. This process enabled the encoder to uncover deep intrinsic data structures, producing an optimized latent feature set that improved classification accuracy and computational efficiency in intrusion detection tasks, as formalized in Eq. (4).

$$\theta^*, \phi^* = \arg \min_{\theta, \phi} \left\| x - g_\phi(f_\theta(x)) \right\|_2^2 \tag{4}$$

where, x is represented as original input feature vector, $f_\theta(x)$ is represented as encoder function parameterized by weights and biases θ; outputs latent vector, $g_\phi(\cdot)$ is represented as decoder function parameterized by weights and biases ϕ; reconstructs input, $\hat{x} = g_\phi(f_\theta(x))$ is represented as reconstructed version of the original input, $\|x - \hat{x}\|_2^2$ is represented as squared L2 reconstruction error measuring how close reconstruction is to the input, θ^*, ϕ^* is represented as optimal encoder and decoder parameters learned during training, Latent dimension (d_h) is represented as size of the encoded hidden representation and Loss function is represented as minimization criterion that guides training.

- Input Feature Vector

The samples of network traffic are referred to as columns that are represented as vectors of all their features. The vector is given a d-dimensional space, which represents some original features of the traffic record [35]. Several samples are combined into a mini-batch to create an efficient training. This batch is then inputted into a DL model where it undergoes forward propagation and an update of parameters. This is expressed in Equation (5).

$$\begin{gathered} x \in \mathbb{R}^d \\ X = \left[x^{(1)}, x^{(2)}, \ldots, x^{(B)}\right] \in \mathbb{R}^{d \times B} \end{gathered} \tag{5}$$

where, x is represented as single input sample representing a network traffic record, $\mathbb{R}^d - \mathrm{d}$ is represented as dimensional real-valued space of features, d is represented as number of original features per sample, B is represented as mini-batch size (number of samples per batch), X is represented as mini-batch matrix containing all input samples and $x^{(i)}$ is represented as the i-th sample in the mini-batch.

- Encoder Transformation

The encoder codes the input vector in a latent representation with several layers. On every layer, the result of the last layer is multiplied by a weight matrix, added to a prejudice vector, and applied to a nonlinear initiation function. *K* layers of this operation are performed to produce the latent vector *h*. The last latent representation represents the most informative properties of the input to be used in processing by the autoencoder. This is expressed in Equation (6).

$$z^{(k)} = W^{(k)}a^{(k-1)} + b^{(k)}$$
$$a^{(k)} = \sigma^{(k)}\left(z^{(k)}\right), k = 1, \ldots, K \quad (6)$$
$$a^{(0)} = x, h \equiv a^{(K)} \in \mathbb{R}^{d_h}$$

where, x is represented as input feature vector (d-dimensional), h is represented as latent representation (d_h-dimensional), $W^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$ is represented as mass matrix of layer k, $b^{(k)} \in \mathbb{R}^{n_k}$ is embodied as bias vector of layer k, $\sigma^{(k)}$ is represented as nonlinear initiation function (ReLU, Sigmoid, Tanh), K is represented as total number of encoder layers and$a^{(k)}$ is represented as activation output of layer k.

- Decoder Reconstruction

The decoder can rebuild the original input of the latent vector based on several layers. In every layer, the weight matrix adds to the input of the previous layer and is combined with a bias vector and multiplied by a nonlinear initiation function [36]. L layers are used to repeat this process to come up with reconstructed output $\hat{x}$. The last output tries to be as similar as the original input, minimizing the reconstruction error and also ensuring that the important information is contained in the latent vector. This is expressed in Equation (7).

$$\tilde{z}^{(\ell)} = \tilde{W}^{(\ell)}\tilde{a}^{(\ell-1)} + \tilde{b}^{(\ell)}$$
$$\tilde{a}^{(\ell)} = \tilde{\sigma}^{(\ell)}\left(\tilde{z}^{(\ell)}\right), \ell = 1, \ldots, L \quad (7)$$
$$\tilde{a}^{(0)} = h, \hat{x} \equiv \tilde{a}^{(L)} \in \mathbb{R}^{d}$$

where, h is represented as latent feature vector from encoder (d_h-dimensional), $\hat{x}$ is represented as reconstructed input vector (d-dimensional), $\tilde{W}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ is represented as mass matrix of decoder layer ℓ, $\tilde{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$ is represented as prejudice vector of decoder layer ℓ, $\bar{\sigma}^{(\ell)}$ is represented as nonlinear activation function (ReLU, Sigmoid, Tanh), L is embodied as total number of decoder layers and $\tilde{a}^{(\ell)}$ is represented as activation output of decoder layer ℓ.

- Reconstruction Loss Calculation

The reconstruction loss is the degree to which the output of the autoencoder $\hat{x}$ approximates the original input x at every sample. Mean Squared Error = squared L2 norm of difference between x and $\hat{x}$. Training this loss to a minimal reduces the parametric encoders and decoders to learn effective latent representations. This is expressed in Equation (8).

$$\ell(x; \theta, \phi) = \|x - \hat{x}\|_2^2 = (x - \hat{x})^\top (x - \hat{x}) \quad (8)$$

where, $x \in \mathbb{R}^d$ is represented as original input vector, $\hat{x} \in \mathbb{R}^d$ is represented as reconstructed output, $\theta = \left\{W^{(k)}, b^{(k)}\right\}_{k=1}^{K}$ is represented as encoder weights and biases, $\phi = \left\{\tilde{W}^{(\ell)}, \tilde{b}^{(\ell)}\right\}_{\ell=1}^{L}$ is represented as decoder weights and biases, K is represented as number of encoder layers and L is represented as number of decoder layers.

- Mini-Batch Loss

To be able to train efficiently, several samples are put in a mini-batch. The mini-batch loss is calculated as the mean of the per-sample reconstruction losses in the batch. This loss is averaged to update the encoder and decoder parameters of the autoencoder through gradient descent. This is expressed in Equation (9).

$$L_{\text{batch}}(X; \theta, \phi) = \frac{1}{B}\sum_{i=1}^{B} \ell\left(x^{(i)}; \theta, \phi\right) \quad (9)$$

where, $X = \left[x^{(1)}, x^{(2)}, \ldots, x^{(B)}\right]$ is represented as mini-batch of input samples, B is represented as mini-batch size, $\ell\left(x^{(i)}; \theta, \phi\right)$ is represented as per-sample reconstruction loss, θ is represented as encoder parameters (weights $W^{(k)}$, biases $b^{(k)}$) and ϕ is represented as decoder parameters (weights $\tilde{W}^{(\ell)}$, biases $\tilde{b}^{(\ell)}$).

- Regularized Objective

L2 regularization is included in mini-batch loss to eliminate overfitting in the training process. It penalizes big weight values in the encoder and decoder networks and promotes the use of simpler models. The regularized objective is the sum of the average reconstruction loss and L2 norms of network parameters. This is expressed in Equation (10).

$$J(X; \theta, \phi) = L_{\text{batch}}(X; \theta, \phi) + \lambda(\|\theta\|_2^2 + \|\phi\|_2^2) \quad (10)$$

where, $L_{\text{batch}}(X; \theta, \phi)$ is represented as mini-batch reconstruction loss, θ is represented as encoder parameters (weights and biases), ϕ is represented as decoder parameters (weights and biases), $\lambda \geq 0$ is represented as regularization coefficient controlling weight decay strength and X is represented as mini-batch of input samples.

- Gradient Computation

The gradients of the regularized objective with respect to encoder and decoder constraints are calculated during the process of training. These gradients determine the manner in which the individual parameters are to be altered towards the path of minimizing loss. To compute these derivatives of each and every layer efficiently, reverse-mode automatic differentiation (backpropagation) is normally employed. This is expressed in Equation (11).

$$g_\theta = \nabla_\theta J(X;\theta,\phi), g_\phi = \nabla_\phi J(X;\theta,\phi) \tag{11}$$

where, g_θ is represented as gradient of objective w.r.t encoder parameters θ, g_ϕ is characterized as gradient of objective w.r.t decoder parameters ϕ, $J(X;\theta,\phi)$ is represented as regularized loss function, θ is represented as encoder weights and biases, ϕ is represented as decoder weights and biases and X is represented as mini-batch of input samples.

- Parameter Update

The Adam optimizer uses adaptive tracking of gradient trends and fluctuations in updating the model parameters. It is a hybrid algorithm that integrates momentum and RMS Prop in order to have rapid convergence and stable updates. This is expressed in Equation (12).

$$\begin{gathered} m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ u_{t+1} = u_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{gathered} \tag{12}$$

where, u_t is represented as current parameter value (weights or biases), g_t is represented as gradient of the parameter at timestep t, m_t, v_t is represented as first and second moment estimates, $\hat{m}_t, \hat{v}_t$ is represented as bias-corrected moment estimates, β_1, β_2 is embodied as exponential deterioration rates for moment estimates (0.9, 0.999), η is represented as learning rate and ϵ is represented as small constant for numerical stability (10^{-8}).

- Iterative Training

The training loop utilizes a mini-batch forward propagation followed by loss calculation and backward gradient followed by updating Adam optimizer parameters. Validation loss is monitored, and early stopping is implemented to avoid overfitting and be able to converge optimally. This is expressed in Equation (13).

$$\begin{gathered} h = f_\theta(x), \hat{x} = g_\phi(h) \\ J(X;\theta,\phi) = \frac{1}{B}\sum_{i=1}^{B} \ell\left(x^{(i)};\theta,\phi\right) \\ g_\theta = \nabla_\theta J(X;\theta,\phi), g_\phi = \nabla_\phi J(X;\theta,\phi) \\ \theta \leftarrow \text{ Adam Update } (\theta, g_\theta), \phi \leftarrow \text{ AdamUpdate } \left(\phi, g_\phi\right) \end{gathered} \tag{13}$$

where, x is represented as input feature vector, h is represented as latent representation from encoder, $\hat{x}$ is represented as reconstructed output from decoder, $J(X;\theta,\phi)$ is represented as mini-batch loss, g_θ, g_ϕ is represented as gradients w.r.t. encoder and decoder parameters, θ, ϕ is represented as learnable masses and biases of encoder and decoder and B is represented as mini-batch size.

- Final Usage

Once autoencoder training has converged, the encoder is then used to produce latent feature vectors of every input. These are denoised and compact representations as the input to the downstream fully connected DNN to classify them. This is expressed in Equation (14).

$$h = f_\theta(x) \in \mathbb{R}^{d_h} \tag{14}$$

where, x is represented as original input feature vector, h is represented as latent feature vector (encoded representation), $f\theta(\cdot)$ is represented as trained encoder function with learned parameters θ and d_h is represented as dimension of the latent space.

The overall procedure of the proposed DL-IDS model is presented in Algorithm 1.

Algorithm 1. Pseudocode: DL-IDS: Deep Learning-Based Intrusion Detection System with Autoencoder and DNN Classifier

Step 1: Data Preprocessing
1. Load the dataset (e.g., UNSW-NB15).
2. Perform one-hot encoding for categorical features:
   - For each categorical variable, create binary vectors.
3. Normalize the numerical features:
   - Apply Min-Max scaling or Z-score normalization to continuous features.
4. Perform feature selection:
   - Use Mutual Information or Recursive Feature Elimination (RFE) to identify the most important features.
5. Split the dataset into training, validation, and test sets.

Step 2: Autoencoder for Feature Extraction (Dimensionality Reduction)
6. Initialize Autoencoder model:
   - Encoder: A series of fully connected layers to reduce the dimensionality of the input features.
   - Decoder: A series of fully connected layers to reconstruct the original input from the encoded representation.
7. Train Autoencoder:
   - Input: Preprocessed dataset with selected features.
   - Loss function: Mean Squared Error (MSE) between the original and reconstructed features.
   - Optimization: Use Adam optimizer to minimize the reconstruction loss.
   - Apply early stopping to avoid overfitting during training.
8. Extract latent features:
   - Use the trained encoder to transform the input data into compact latent vectors.

Step 3: Train the DNN Classifier
9. Initialize Deep Neural Network (DNN) classifier:
   - Input: Latent features from the Autoencoder.
   - Architecture: Fully connected layers with ReLU activation functions.
   - Output: Softmax layer to classify into multiple classes (normal/malicious).
10. Train DNN classifier:
   - Loss function: Cross-Entropy Loss (for multi-class classification).
   - Optimization: Adam optimizer to minimize classification loss.
   - Apply early stopping based on validation loss.

Step 4: Model Evaluation
11. Evaluate the trained DNN classifier on the test set:
   - Compute accuracy, precision, recall, F1-score using confusion matrix.
12. Report the performance metrics (Accuracy, Precision, Recall, F1-Score).
13. Optionally, perform robustness testing:
   - Evaluate the model on adversarial attacks or noisy data.
   - Test performance on different datasets (e.g., CICIDS2017, BoT-IoT).

Step 5: Deployment (Real-time Detection in Cloud)
14. Deploy the trained model to the cloud environment:
   - Integrate the model with real-time traffic monitoring systems.
   - Classify incoming network traffic as normal or malicious in real-time.
15. Continuously monitor and update the model as needed based on new attack patterns.

3.4. Dataset Splitting

Once the data are cleaned and features are extracted, they are divided into separate sets for training, validation, and testing. Typically, the largest portion is devoted to training the model, validation, which is smaller in size, is applied to tweak hyperparameters and monitor overfitting, and the remainder is dedicated to testing and assessing final model performance. This is how the distribution model can extrapolate extremely fine to the data

which it has never seen, as well as provide reliable performance metrics on different subsets. This is reflected in Equation (15).

$$D = D_{\text{train}} \cup D_{\text{val}} \cup D_{\text{test}}, D_{\text{train}} \cap D_{\text{val}} = D_{\text{train}} \cap D_{\text{test}} = D_{\text{val}} \cap D_{\text{test}} = \emptyset \quad (15)$$

D is the full dataset that has gone through preprocessing and feature extraction, D_{train} is dataset used for model learning, D_{val} is dataset used for hyperparameter tuning and early stopping as well as D_{test} is the dataset for final evaluation of the model performance.

3.5. Model Design Using DNN

A fully connected DNN architecture was designed for cloud intrusion detection, as illustrated in the proposed model by Nippatla et al. (2024) In this framework, features extracted using an autoencoder were fed into multiple dense layers, enabling the network to learn complex, non-linear relationships within the data. The initial layers captured low-level patterns associated with normal and malicious cloud activities, while deeper layers identified higher-level abstractions. ReLU activation functions were employed to accelerate learning, and dropout was applied to mitigate overfitting. The output layer classified network traffic into attack categories using the UNSW-NB15 dataset, thereby supporting accurate and reliable threat detection across diverse intrusion scenarios [37].

This chart shows a DNN that is fully connected, and it is used in the detection of intrusion with the assistance of the UNSW-NB15 dataset. The network begins with an input layer that receives features that are extracted and processed from the network traffic. The inputs are then processed in three hidden layers, with each neuron of one layer connected to all neurons of the next layer; this enables the interactions between the features to be modeled in a very intricate manner. Nonlinearly distinguishing and, therefore, more precisely describing the normal and malicious behaviors are done with nonlinear activations, such as ReLU. Lastly, the output layer provides probabilities of ten classes that include normal, fuzzers, DoS, exploits, shellcode, and worms, among other types of attacks, and this makes it possible to classify the network intrusions into several classes with high accuracy.

- Input Layer

The feature vector that results after the preprocessing of the autoencoders is fed into the input layer. All input samples are coded in a column vector with all the features. This is the beginning point of spreading the information in the neural network. This is expressed in Equation (16).

$$\mathrm{x} \in \mathbb{R}^d \quad (16)$$

where, x is represented as input feature vector, d is represented as number of input features and $\mathbb{R}$ is represented as the field of real numbers; all components of the input vector x are real valued.

- First Hidden Layer

In the first hidden layer, input vector is multiplied by weight matrix and added to a bias vector to compute pre-activation values. Non-linear activation function, such as ReLU, is then applied to announce nonlinearity and enable network to learn complex patterns. Output of this layer becomes the input for the next hidden layer. This is expressed in Equation (17).

$$z^{(1)} = W^{(1)}x + b^{(1)}, a^{(1)} = \sigma\left(z^{(1)}\right) \quad (17)$$

where, $x \in \mathbb{R}^d$ is represented as input feature vector, $W^{(1)} \in \mathbb{R}^{n_1 \times d}$ is represented as weight matrix for first hidden layer, $b^{(1)} \in \mathbb{R}^{n_1}$ is characterized as Bias vector, n_1 is signified as number of neurons in the first hidden layer and $\sigma(\cdot)$ is represented as initiation function.

- Second Hidden Layer

Second hidden layer receives the activation outputs from first hidden layer as its input. Each neuron computes a weighted sum during its operation and a bias value is incorporated. The output activations of this layer are produced by the application of a non-linear activation function, for example ReLU, which enables the network to represent complicated patterns in the data. This is demonstrated in equation (18).

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}, a^{(2)} = \sigma\left(z^{(2)}\right) \quad (18)$$

where, $a^{(1)} \in \mathbb{R}^{n_1}$ this model is represented by the output vector from first hidden layer, $W^{(2)} \in \mathbb{R}^{n_2 \times n_1}$ is considered as a mass matrix of second hidden layer., mapping n_1 inputs to n_2 neurons, $b^{(2)} \in \mathbb{R}^{n_2}$ represents the bias vector for an input of second hidden layer, n_2 is represented as number of neurons in second hidden layer and $\sigma(\cdot)$ is represented as activation function applied element-wise.

- Third Hidden Layer

Third hidden layer takes the activation outputs from second hidden layer as input. Weighted sum of inputs is computed by each neuron, then a bias term is added and a non-linear activation function like ReLU is useful. Through this, network is able to discern the data's more abstract patterns that it then feeds to the output layer. This is expressed in Equation (19).

$$z^{(3)} = W^{(3)}a^{(2)} + b^{(3)}, a^{(3)} = \sigma\left(z^{(3)}\right) \tag{19}$$

where, $a^{(2)} \in \mathbb{R}^{n_2}$ is represented as output vector from the second hidden layer, $W^{(3)} \in \mathbb{R}^{n_3 \times n_2}$ is denoted as mass matrix of third hidden layer, mapping n_2 inputs to n_3 neurons, $b^{(3)} \in \mathbb{R}^{n_3}$ is represented as bias vector for third hidden layer, n_3 is characterized as number of neurons in third hidden layer and $\sigma(\cdot)$ is signified as activation function applied element-wise.

- Output Layer

The output layer computes the logits by performing a linear combination of third hidden layer activations and output weights, adding bias term. These logits are then transformed into class probabilities using softmax function, enabling multi-class classification. Each probability corresponds to a exact attack type or else normal traffic, allowing model to predict most likely class for a given input. This is expressed in Equation (20).

$$z^{(4)} = W^{(4)}a^{(3)} + b^{(4)}, \hat{y}_k = \frac{e^{z_k^{(4)}}}{\sum_{j=1}^{n_4} e^{z_j^{(4)}}}, k = 1, \ldots, n_4 \tag{20}$$

where, $a^{(3)} \in \mathbb{R}^{n_3}$ is represented as activation output from the third hidden layer, $W^{(4)} \in \mathbb{R}^{n_4 \times n_3}$ is characterized as mass matrix connecting third hidden layer to output layer, $b^{(4)} \in \mathbb{R}^{n_4}$ is signified as bias vector for output layer, $n_4 = 10$ is represented as number of output classes: Normal, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms also $\hat{y}_k$ is represented as predicted probability for class k.

- Prediction

The prediction step determines the class of an input by selecting one with highest probability from softmax output. This converts the probability distribution into a discrete class label, representing the most likely category of the input traffic. The resulting predicted class can then be used for evaluation or further processing in the IDS. This is expressed in Equation (21).

$$\hat{c} = \arg\max_k \hat{y}_k \tag{21}$$

where, $\hat{y}_k$ is represented as softmax probability for class k and $\hat{c}$ is represented as predicted class label (index of the maximum probability).

3.6. Cloud Intrusion Detection

DL-IDS on Cloud Infrastructure Security comprises several security layers to identify as well as classify cyber threats in real-time. The initial phase of the system is to collect logs and monitor the data flow route in real-time with the aim of better understanding what the cloud is up to. The idea is to follow the signatures of the potential attacks to those that are already identified by the big data using threat intelligence reasoning [38]. In addition to the DNN, an autoencoder is utilized with the aim of performing the anomaly detection, which will be capable of pointing at the behavior that is not normal. Finally, the identified intrusions are categorized using the categories of the UNSW-NB15 attacks to identify the exact threat. This approach is based on the combination of behavior-based, signature-based, and DL approaches that lead to accurate, adaptive, and pervasive cloud intrusion detection.

3.6.1. Log Collection

The system has the ability to extract non-processed cloud events from various sources including firewalls, virtual machines (VMs), containers, and API gateways. The events gathered in this manner are merged and kept in one common dataset which is going to be used for preprocessing and analyzing. This common dataset is the input for the next stages of feature extraction and detection. This is expressed in Equation (22).

$$\mathcal{R} = \{r_1, r_2, \ldots, r_N\} \tag{22}$$

where, $\mathcal{R}$ is represented as complete set of collected cloud records, r_i is represented as the i^{th} raw event/log entry from any cloud source and N is represented as total number of collected records.

3.6.2. Traffic Monitoring & Feature Extraction

The system monitors cloud network traffic and extracts meaningful attributes from packets or flow windows. Each raw record or window is changed into a fixed length numerical feature vector suitable for model input. This feature representation enables ML models to process traffic consistently and detect anomalies. This is expressed in Equation (23).

$$x = \Phi(r, W) \in \mathbb{R}^d \tag{23}$$

where, r is represented as a raw log/packet record from cloud monitoring sources, W is represented as a window or batch of multiple sequential packets/records, $\Phi(\cdot)$ is represented as feature extraction function that transforms raw data into numeric features, $x \in \mathbb{R}^d$ is represented as output feature vector after extraction and d is represented as dimensionality of the feature vector (number of extracted features).

3.6.3. Threat-Intelligence Matching

Threat-intelligence matching checks whether the extracted feature vector x contains elements that match any known attack indicators in the set I. A matching function $M(x, I)$ evaluates this relationship using exact or fuzzy similarity rules. The result is a binary threat flag (or confidence score) indicating whether the record is associated with a known malicious signature. This is expressed in Equation (24).

$$\text{flag}_{TI} = M(x, I) \in \{0,1\} \tag{24}$$

where, $x \in \mathbb{R}^d$ is represented as feature vector for a network record or flow, I is represented as set of known threat indicators (e.g., malicious IPs, domains, hashes, signatures), $M(x, I)$ is represented as matching function (exact match, similarity scoring, or fuzzy matching) and flag $_{TI}$ is represented as output threat flag ($0 =$ no match, $1 =$ known threat; or a score in $[0,1]$).

3.6.4. Anomaly Detection

The autoencoder detects anomalies by first encoding the input feature vector x into a compact latent representation h. It then reconstructs an approximation $\hat{x}$, capturing the usual data patterns learned during training. Reconstruction error $r(x)$ serves as the anomaly score-larger values indicate abnormal or suspicious behavior. This is expressed in Equation (25).

$$r(x) = \left\| x - g_\phi(f_\theta(x)) \right\|_2^2 \tag{25}$$

where, $x \in \mathbb{R}^d$ is represented as input feature vector, $f_\theta(\cdot)$ is represented as encoder function through learnable parameters θ, $g_\phi(\cdot)$ is characterized as decoder function through learnable parameters ϕ, $\hat{x} = g_\phi(f_\theta(x))$ is represented as reconstructed version of input, $r(x)$ is represented as anomaly score (reconstruction error) and $\| \cdot \|_{\frac{1}{2}}^2$ is represented as squared L2 norm used to measure reconstruction difference.

3.6.5. Attack Classification

The selected features (latent from autoencoder or original) are fed into a fully connected DNN. DNN computes logits using weights and biases, then applies the softmax function to produce class probabilities over UNSW-NB15 attack categories. Predicted class corresponds to index with highest probability. This is expressed in Equation (26).

$$\hat{c} = \arg\max_k \frac{\exp\left((W_{\text{DNN}} a + b_{\text{DNN}})_k\right)}{\sum_j \exp\left((W_{\text{DNN}} a + b_{\text{DNN}})_j\right)} \tag{26}$$

where, $a \in \mathbb{R}^{d_h}$ is represented as input feature vector to the DNN (latent or selected features), $W_{\text{DNN}} \in \mathbb{R}^{n_{\text{claseas}} \times d_h}$ is represented as weight matrix of the DNN, $b_{\text{DNN}} \in \mathbb{R}^{n_{\text{clasase}}}$ is represented as bias vector of the DNN, $\hat{y}_k$ is represented as predicted probability for class k, $\hat{c}$ is represented as predicted class label (argmax of softmax output) and $n_{\text{classes}} = 10$ is represented as number of UNSW-NB15 attack categories.

3.6.6. Decision & Thresholds

The decision module raises an alert when the system detects suspicious activity. An alert is triggered if anomaly score from the autoencoder exceeds a predefined verge, if threat-intelligence matching flags a known threat, or if the classifier predicts an attack class. Otherwise, the traffic is marked as normal, ensuring accurate detection while minimizing false positives. This is expressed in Equation (27).

$$\text{Alert } = \begin{cases} 1, & \text{if } r(x) > \tau \vee \text{ flag }_{TI} = 1 \vee \hat{c} \in A \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

where, $r(x)$ is represented as anomaly score computed from autoencoder reconstruction error, τ is represented as predefined threshold for anomaly detection, flag $_{TI}$ is represented as threat-intelligence matching flag (1 if matched, 0 otherwise), $\hat{c}$ is represented as predicted class label from the classifier and A is represented as set of indices representing attack classes.

3.6.7. Training & Optimization

The autoencoder and DNN are trained using mini-batch optimization with the Adam optimizer. The autoencoder minimizes reconstruction loss to learn compact latent representations, while the DNN minimizes cross-entropy loss for accurate classification. Parameter updates are performed using Adam, and initial stopping based on authentication loss prevents overfitting, ensuring robust generalization. This is expressed in Equation (28).

$$\text{Parameters }^{*} = \arg\min_{\theta,\phi} \frac{1}{B}\sum_{i=1}^{B} \left\| x^{(i)} - g_{\phi}\left(f_{\theta}\left(x^{(i)}\right)\right)\right\|_2^2, \theta \leftarrow \text{AdamUpdate}(\theta, \nabla_{\theta} L) \tag{28}$$

where, $x^{(i)}$ is represented as input feature vector for sample i, $f_{\theta}(\cdot), g_{\phi}(\cdot)$ is represented as encoder and decoder functions with parameters θ, ϕ, B is represented as mini-batch size, $\nabla_{\theta} L$ is represented as gradient of loss w.r.t. parameters θ, θ, ϕ is signified as trainable parameters of the autoencoder and Adam Update is represented as optimization step using Adam algorithm.

3.7. Model Training Using Adam Optimization

Model training was carried out using the Adam optimizer, as described by Srinivasan et al. (2024) which adaptively adjusted the learning rate through estimates of the first and second moments of the gradients, as illustrated in the optimization flowchart in Figure 2. The weight update process employed bias-corrected moment terms to ensure stable and efficient convergence. Cross-entropy loss guided learning by comparing predicted probabilities with true labels, while continuous parameter adjustments minimized this loss and improved accuracy [39]. Model performance was monitored using validation loss at each training step, and early stopping was applied when no further improvement was observed. This strategy reduced overfitting and enhanced the model's generalization capability on unseen data.

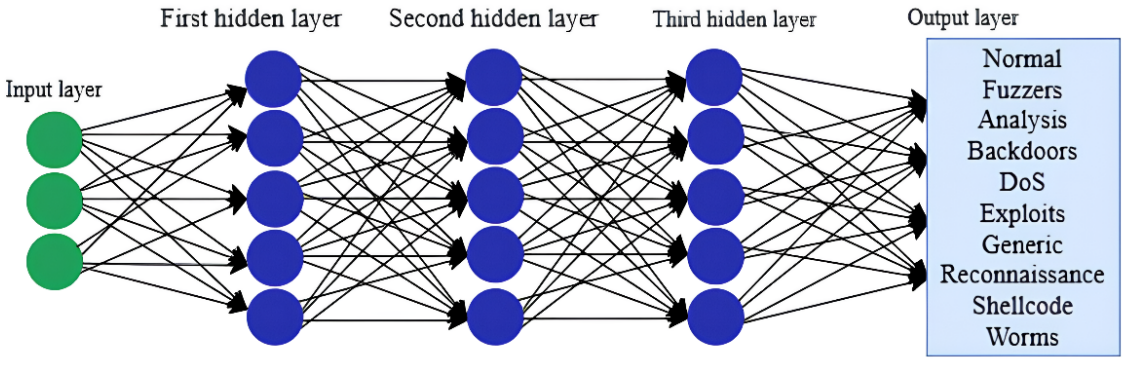

Figure 2. Architecture diagram of DNN.

The flowchart demonstrates the process of training a DL model that uses the Adam optimizer and early stopping in Figure 3. The initial one is to establish the model and prepare the input data, and subsequently, the network is executed to get the predictions through a forward pass. The loss in cross-entropy is determined using the true and predicted labels. Then, the parameters of the model are modified based on the Adam maximization algorithm that relies on the previously calculated loss gradients. The validation performance is checked after every

repetition; in case the validation loss is improved, training is further continued; otherwise, early stopping is used. After achieving the stopping criterion, the final trained model is provided, and this is ensured to be the best-performing model without overfitting.

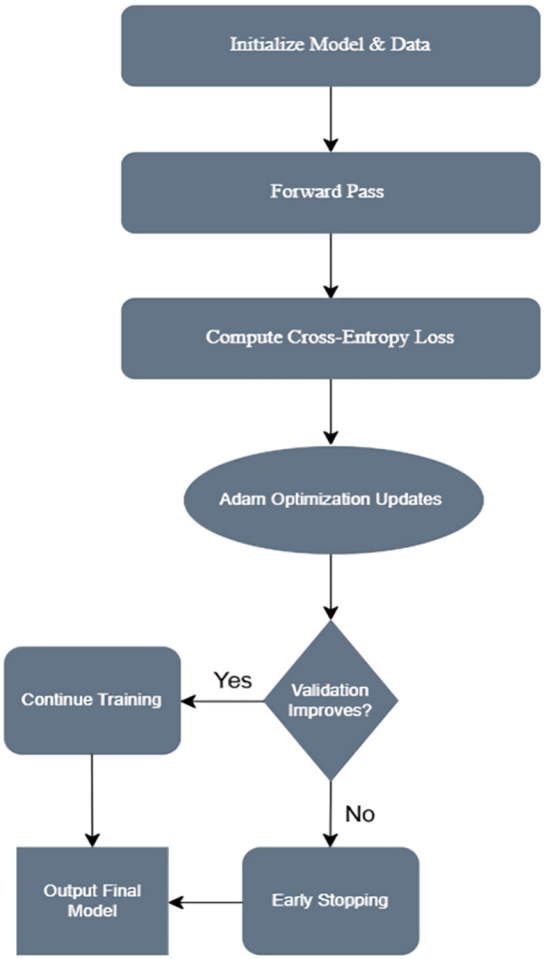

Figure 3. Flowchart for Adam Optimization.

- Initialize Model & Data

Model training is initiated with the process of setting all trainable parameters, weights and biases, for every layer in the network to random initial values. After that, the training and validation datasets are prepared for further processing. The network is thus enabled to perform forward and backward passes during training. This is mathematically represented in Equation (29).

$$\theta = \left\{W^{(\ell)}, b^{(\ell)}\right\}_{\ell=1}^{L} \tag{29}$$

where, θ is represented as set of all trainable parameters, $W^{(\ell)}$ is represented as mass matrix of layer ℓ, $b^{(\ell)}$ is represented as bias vector of layer ℓ and L is represented as entire number of layers in network.

- Forward Pass

In the forward pass, input features are propagated through each hidden layer, computing activations using weights, biases, and a non-linear activation function. The final layer calculates logits, which are converted into probabilities via softmax for multi-class prediction. This process produces the predicted outputs for each input sample. This is expressed in Equation (30).

$$a^{(\ell)} = \sigma\left(W^{(\ell)}a^{(\ell-1)} + b^{(\ell)}\right), y^{k} = \frac{e^{z_k}}{\sum_j e^{z_j}}, z = W^{(L)}a^{(L-1)} + b^{(L)} \tag{30}$$

where, $a^{(\ell)}$ is represented as activation output of layer ℓ, σ is represented as activation function (e.g., ReLU), y^{k} is represented as predicted probability for class k, $W^{(\ell)}, b^{(\ell)}$ is represented as weight and bias of layer ℓ and L is characterized as total number of layers.

- Compute Cross-Entropy Loss

Cross-entropy loss measures change between the predicted probabilities and true labels for multi-class classification [40]. It penalizes incorrect predictions more heavily and guides the network during training. Minimizing this loss helps the model improve prediction accuracy over the training data. This is expressed in Equation (31).

$$L_{CE} = -\sum_{k} y_k \log \hat{y}_k \tag{31}$$

where, y_k is represented as true label for class k (one-hot encoded), $\hat{y}_k$ is represented as predicted probability for class k and L_{CE} is represented as cross-entropy loss value.

- Adam Optimization Updates

Adam optimizer updates model parameters by combining gradients with adaptive estimates of first and second moments, allowing for faster and more stable convergence. Each parameter is adjusted in proportion to its gradient while accounting for past updates. This helps prevent oscillations and improves learning efficiency across all layers. This is expressed in Equation (32).

$$\theta \leftarrow \theta - \eta \cdot \text{ AdamStep } (\nabla_\theta L_{CE}) \tag{32}$$

where, θ is represented as trainable parameters (weights and biases), η is represented as learning rate controlling update magnitude, $\nabla_\theta L_{CE}$ is represented as gradient of the cross-entropy loss w.r.t θ and Adam Step(.) is represented as function computing Adam's adaptive update using moment estimates.

- Validation Check

After each training epoch, model is assessed on a separate validation set to measure its generalization performance [41]. The validation loss is monitored to detect improvements; if the loss decreases, training continues. If no development is observed for a set number of aeras, early stopping is triggered to prevent overfitting. This is expressed in Equation (33).

$$\text{Stop if } L_{\text{val}}^{(t)} \geq L_{\text{val}}^{(t-1)} \text{ for } p \text{ consecutive epochs} \tag{33}$$

where, $L_{\text{val}}^{(t)}$ is represented as validation loss at epoch t, p is represented as patience, number of epochs to wait before stopping, t is represented as current epoch index.

- Continue Training/Early Stopping

The model training proceeds epoch by epoch, updating parameters using the optimizer while monitoring validation performance. If the validation loss improves, training continues; otherwise, early stopping stops training to prevent overfitting. This technique guarantees that model is capable of making good predictions for new data that model has not been trained on and whole process of drill the model is not too long. This is mathematically formulated in Equation (34).

$$\text{Stop Training if } L_{\text{val}}^{(t)} \geq L_{\text{val}}^{(t-1)} \text{ for } p \text{ consecutive epochs} \tag{34}$$

where, $L_{\text{val}}^{(t)}$ is characterized as validation loss at epoch t, p is signified as patience, number of epochs to wait before stopping and t is denoted as current epoch index.

- Output Final Model

After the model undergoes training and early stopping, it selects and keeps the parameters that yielded the least value for the loss function. The parameters that have been optimized are then stored and are later used to brand predictions on fresh data. Deployed model thus incorporates patterns learned during the training stage. This is represented in Equation (35).

$$\theta^* = \arg\min_{\theta} L_{\text{CE}} \tag{35}$$

where, θ^* is characterized as final trained model parameters (weights and biases) and L_{CE} is embodied as cross-entropy loss used throughout training.

4. Result and Discussion

The results of experimental activities confirm the idea that the proposed DL-IDS is superior not only regarding the detection capability but also compared to the traditional methods. The accuracy of the system is

99.12%, where the accuracy is shown as the system can easily and reliably distinguish between normal and malignant traffic. The 98.87% accuracy confirms its ability to reduce the cases of the false positives and, therefore, reduce the number of alerts that are not necessary to the administrators. Its potency of drawing about 99.54% of the intrusion attempts is an indicator that it is quite important in the prevention of undetected intrusions. The F1-score is 99.21%, resulting in a well-balanced trade-off between the precision and the recall, hence the reliability and the efficiency. It is this overlap of metrics that ends up persuading one of the effectiveness of the preprocessing pipeline and also of the autoencoder-based feature extraction in the accuracy increase in the classification. They also mention the scalability of the framework and hence its ability to run in dynamic cloud environments where the amount of traffic is high. With the literature benchmarking, the proposed system still exhibits increased adaptability and robustness; therefore, it can be considered a viable solution to the real cloud security applications.

4.1. Confusion Matrix

The performance of the IDS on the binary classification task is illustrated in Figure 4. Out of the total predictions, 355 normal instances were correctly classified as normal (true positives), while 6 normal instances were misclassified as anomalies (false negatives). Similarly, 434 anomalous instances were correctly identified as anomalies (true negatives), and only 5 anomalous instances were incorrectly classified as normal (false positives). These results indicate that the proposed DL–based cloud IDS achieved very high accuracy in distinguishing between normal and anomalous traffic, with minimal misclassification. Overall, the classification performance strongly demonstrated the effectiveness and reliability of the proposed approach, consistent with the evaluation practices reported by Dyavani et al. (2024) [42].

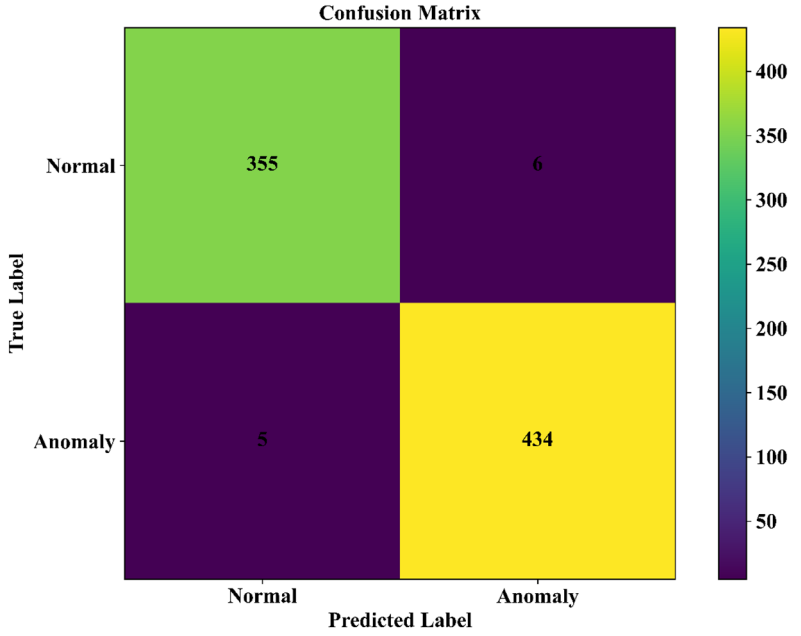

Figure 4. Confusion Matrix.

4.2. DNN Classification Accuracy

The Figure 5 shows the results of the validation and training during the 40 epochs. The training accuracy is initially determined to be approximately 0.86, and it progressively reaches the mark of approximately 0.98 after epoch 39 [43]. Instead, the validation accuracy begins at approximately 0.90 and shows slight fluctuations, which are caused by changes in batches, and approaches approximately 0.98 over the final epochs, thereby proving good generalization. All the two curves are consistent in their upward movement, which depicts good learning as well as a slight overfitting. The narrowness of the training and validation accuracies implies that not only has the model learned the underlying patterns in the UNSW-NB15 dataset, but it has also kept the performance of the model on unseen data consistent.

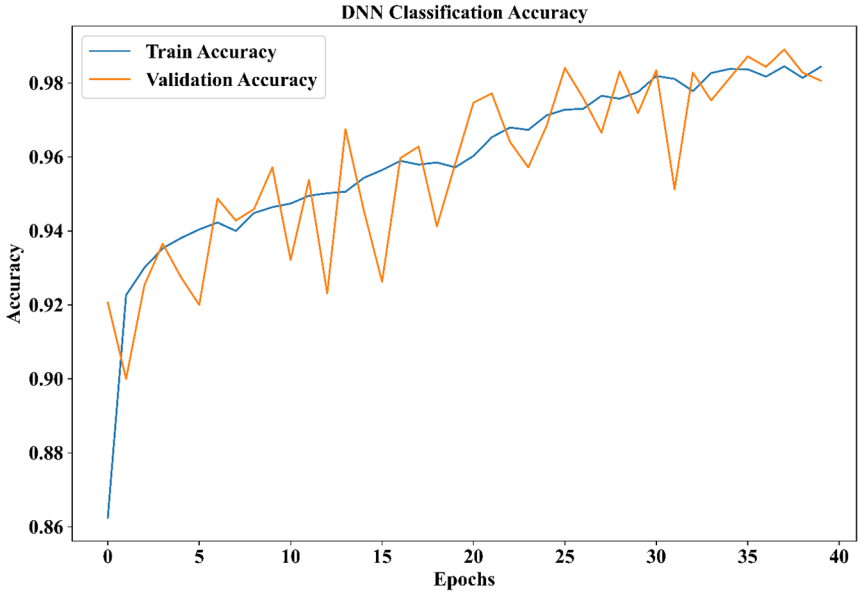

Figure 5. DNN Classification Accuracy.

4.3. Autoencoder Training Loss

Figure 6 shows the Training and validation mean squared error (MSE) characterization in the graph is through 29 epochs. Initially, the training loss is approximately 0.035, dropping rapidly to approximately 0.003 in the 2nd epoch, and the validation loss is 0.005 at the start and then drops in a similar fashion. The training and validation losses are approximately 0.001 at epoch 5 and do not increase even in the subsequent epochs and are still low. The rapid convergence indicates that the input features are well learned and rebuilt. The closeness of the training and validation losses is indicative of the fact that the autoencoder adapts to new data very well and, therefore, prevents overfitting and guarantees that the anomaly detection procedure is accurate.

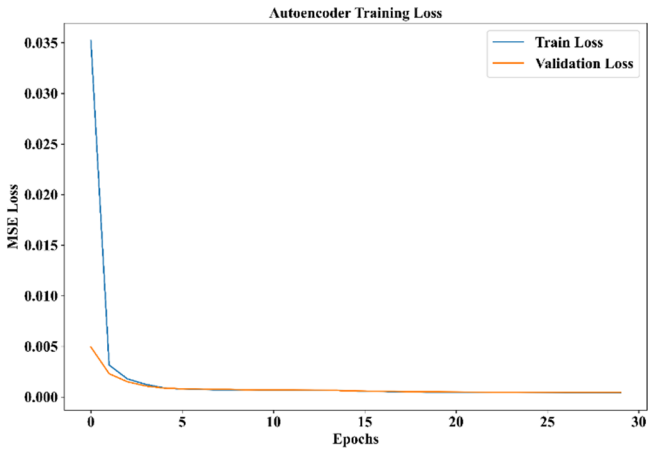

Figure 6. Autoencoder Training Loss.

4.4. Feature Importance

In Figure 7 the 16 network features have their contribution to intrusion detection illustrated in the graph. The feature dttl takes the first place with the importance score of nearly 0.19 and the second place with the score of 0.16; they are the ones that contribute significantly to model predictions. The scores of states and sttl are approximately 0.075, and service and ct_dst_src_ltm take the values of about 0.058. The not-very-high contributors include proto, ct_state_ttl, dwin, and swin, with the scores of 0.02-0.025. The features with the lowest scores are dload, dmean, ct_dst_ltm, ct_srv_src, and sinpkt, with scores of less than 0.01, indicating their insignificance. These rankings are useful in the selection of features and model interpretability.

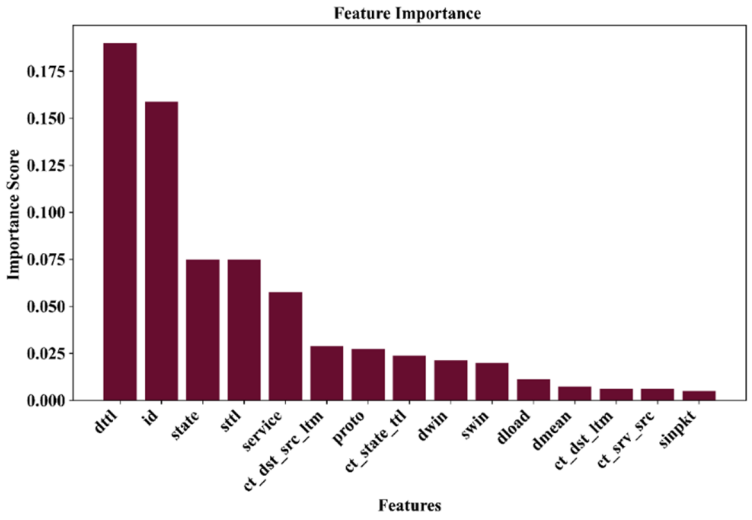

Figure 7. Feature Importance.

4.5. FNR & FPR

Figure 8 defines the performance with regard to errors of the developed IDS. False positive rate (FPR) has been maintained at 0.0166, which implies that the frequency of regular cases that have been improperly termed as attacks is very low [44]. False negative rate (FNR) is slightly lower at 0.0114, which represents a minute percentage of the attack instances that the system fails to detect. The minimal values show that the model is highly dependable and strong in the recognition and separation of normal and abnormal cloud traffic. The graph indicates that the two types of classification errors are minimal, and therefore the capacity of the DNN and autoencoder-based architecture to identify intrusions is prolific.

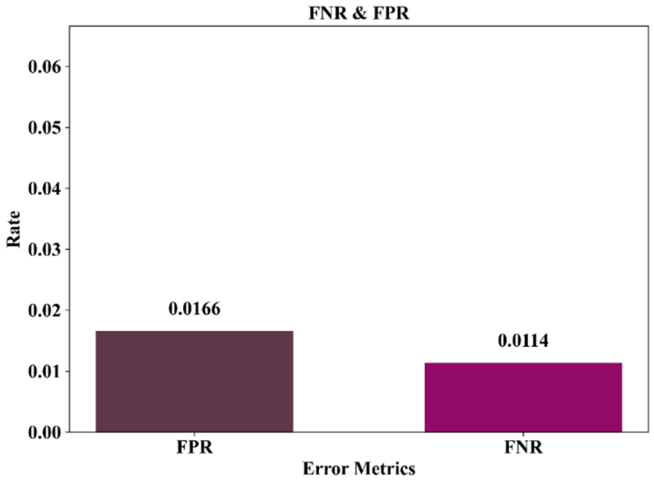

Figure 8. FNR & FPR.

4.6. Overall Performance Metrics

As illustrated in Figure 9 the evaluation of the model is represented through the bar chart using four important metrics. First of all, Accuracy is very high 99.12%, which means that hardly any wrong predictions were made in the first place. Then Precision, which measures the same thing as Accuracy but a bit differently, comes second with 98.87% representing the model's capacity of avoiding false positives. Recall goes up to 99.54%, which is a sign of great sensitivity in discovering exact positives. The F1-score, a metric that syndicates both precision and recall, is 99.21%, thus verifying very good performance in all aspects. The numbers of all metrics are at least 98.8%, with Recall being the highest one, which indicates that the model is especially good at gathering the relevant examples. The y-axis values go from 0.95 to 1.00, which reflects the high scores that are constantly present in all the evaluation parameters.

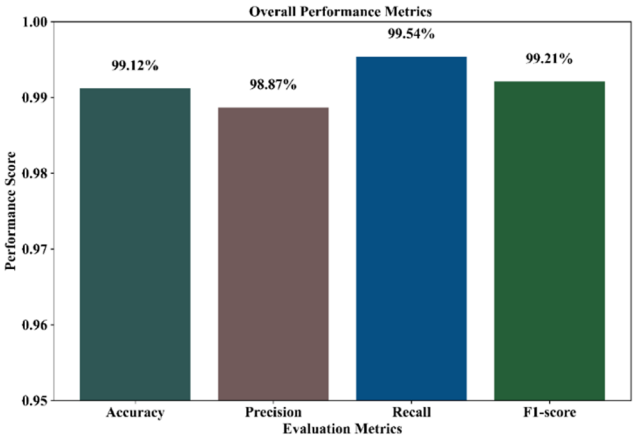

Figure 9. Overall Performance Metrics.

4.7. ROC Curve for Intrusion Detection

As illustrated in Figure 10, the ROC curve for intrusion detection demonstrated outstanding performance, consistent with the evaluation methodology reported by Ramar et al. (2020). The curve lay very close to the upper-left corner, indicating an excellent balance between true positive rate and false positive rate. The model's performance was further quantified by an Area Under the Curve (AUC) value of 0.9991, which is nearly perfect. The blue ROC curve clearly showed the model's superior ability to distinguish between intrusion and normal activity when compared with the orange dashed line representing a random classifier. Such a high AUC value highlighted the model's reliability for cybersecurity applications, as it accurately identified threats while maintaining minimal false alarms. This is a highly advantageous aspect for systems that demand real-time monitoring and prevention of intrusions [45].

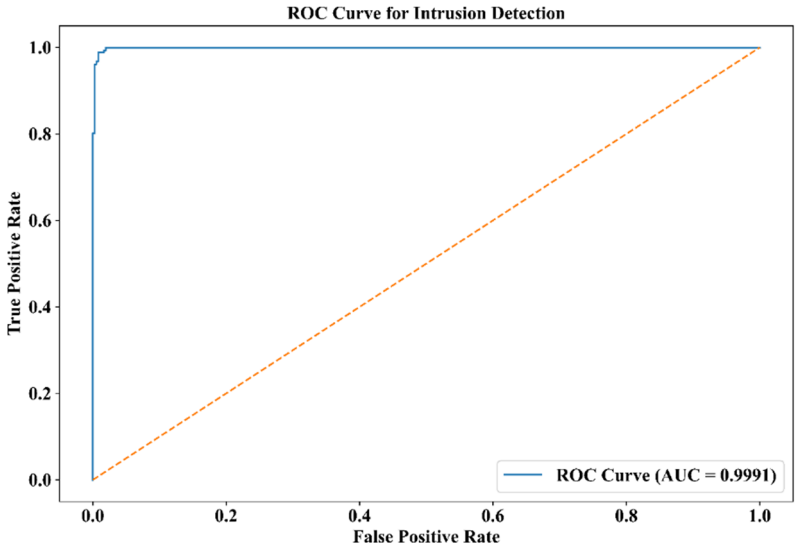

Figure 10. ROC Curve for Intrusion Detection.

Table 1 compares the performance of various intrusion detection models based on accuracy, precision, recall, F1-score, training time, and prediction time. Among the models, Random Forest (RF) and XGBoost demonstrate strong performance with relatively lower training time and prediction time. However, deep learning models like LSTM and the proposed DL-IDS method show the highest accuracy, precision, and recall (99.12% and 99.54%, respectively), though they require longer training times (~20 h) and slightly higher prediction time (~1 s). The CNN and GRU models offer a balance between performance and efficiency, with GRU having a faster training time than LSTM but slightly lower recall.

Table 1. Comparison Table.

Model	Accuracy	Precision	Recall	F1-Score	Training Time (h)	Prediction Time (s)
Random Forest [46]	98%	98%	97%	97%	~2 h	~0.1 s
Support Vector Machine [47]	95%	94%	96%	95%	~6 h	~0.15 s
Convolutional Neural Network [48]	98%	98%	97%	97%	~15 h	~0.5 s
Long Short-Term Memory [49]	99.12%	98.87%	99.54%	99.21%	~20 h	~1 s
Gated Recurrent Units [50]	98%	97%	96%	96%	~18 h	~0.8 s
XGBoost [51]	97%	97%	96%	96%	~5 h	~0.2 s
Proposed Method (DL-IDS)	99.12%	98.87%	99.54%	99.21%	~20 h	~1 s

4.8. Ablation Study

Table 2 compares the performance of different model versions used in the proposed DL-IDS. It shows that the full model, which incorporates both autoencoder-based feature extraction and feature selection, achieves the highest accuracy (99.12%), precision (98.87%), recall (99.54%), and F1-score (99.21%). The results highlight the importance of both feature extraction and selection in improving the model's performance.

Table 2. Ablation study table.

Ablation Study Component	Model Version	Accuracy	Precision	Recall	F1-Score
Basic Model (No Autoencoder, No Feature Selection)	DNN without AE or Feature Selection	95.20%	94.50%	96.80%	95.60%
Model with Autoencoder (No Feature Selection)	DNN + AE	98.15%	97.70%	98.25%	97.97%
Model with Feature Selection (No Autoencoder)	DNN + Feature Selection	97.50%	96.80%	98.00%	97.39%
Full Model (Autoencoder + Feature Selection)	Proposed DL-IDS (Full Model)	99.12%	98.87%	99.54%	99.21%

5. Conclusions & Future Scope

The study reveals that intrusion detection based on DL has potential to build up cloud infrastructure security to a considerable extent. When using back-end processes, autoencoder aimed at feature extraction, also DNN classifier, proposed DL-IDS achieves remarkable results of 99.12% accuracy, 98.87% precision, 99.54% recall, and 99.21% F1-score, thus, outperforming many of traditional and ML-based systems. The system well maintains the no false alarminess required for highly sensitive detections, thus, it is suitable for real-time deployments in cloud environments. Use of the UNSW-NB15 dataset enables the actual training scenarios while the integration of normalization, one-hot encoding, and feature selection improves efficiency and prevents overfitting. However, there are still some difficulties like the requirement of huge labeled datasets, the issue of computational above, and the susceptibility to adversarial attacks. In future research, plan is to combine federated learning for the protection of data privacy, lightweight architectures for resource-constrained environments, and adversarial manipulations resistance enhancement. Besides, the application of hybrid methods that merge DL with bio-inspired optimization and explainable AI will be considered to boost interpretability and adaptability. These future endeavors will ensure that DL-IDS keeps pace with the evolving threats and thus, will be the provider of sustainable and reliable cloud security solutions for organizations increasingly operating in complex digital ecosystems.

Funding

This research received no external funding.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

The data used in this study are publicly available from the UNSW-NB15 dataset. The dataset can be accessed from: https://research.unsw.edu.au/projects/unsw-nb15-dataset. No new data were created in this study. Further processed data can be made available by the author upon reasonable request.

Conflicts of Interest

The author declares no conflict of interest.

Use of AI and AI-Assisted Technologies

During the preparation of this work, the author used AI-assisted tools for language refinement and proofreading purposes only. After using these tools, the author reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

1. Karkazis, P.A.; Railis, K.; Prekas, S.; et al. Intelligent Network Service Optimization in the Context of 5G/NFV. *Signals* **2022**, *3*, 587–610.
2. Li, Y.; Hu, H.; Liu, W.; et al. An Optimal Active Defensive Security Framework for the Container-Based Cloud with Deep Reinforcement Learning. *Electronics* **2023**, *12*, 1598.
3. Wang, Y.; Dong, S.; Fan, W. Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing. *Mathematics* **2023**, *11*, 3364.
4. Gautam, M. Deep Reinforcement Learning for Resilient Power and Energy Systems: Progress, Prospects, and Future Avenues. *Electricity* **2023**, *4*, 336–380.
5. Isong, B.; Kgote, O.; Abu-Mahfouz, A. Insights into Modern Intrusion Detection Strategies for Internet of Things Ecosystems. *Electronics* **2024**, *13*, 2370.
6. Alqahtani, F.; Almutairi, M.; Sheldon, F.T. Cloud Security Using Fine-Grained Efficient Information Flow Tracking. *Future Internet* **2024**, *16*, 110.
7. Uszko, K.; Kasprzyk, M.; Natkaniec, M.; et al. Rule-Based System with Machine Learning Support for Detecting Anomalies in 5G WLANs. *Electronics* **2023**, *12*, 2355.
8. Chauhan, M.; Shiaeles, S. An Analysis of Cloud Security Frameworks, Problems and Proposed Solutions. *Network* **2023**, *3*, 422–450.
9. Aziz, Z.; Bestak, R. Insight into Anomaly Detection and Prediction and Mobile Network Security Enhancement Leveraging K-Means Clustering on Call Detail Records. *Sensors* **2024**, *24*, 1716.
10. Beshah, Y.K.; Abebe, S.L.; Melaku, H.M. Drift Adaptive Online DDoS Attack Detection Framework for IoT System. *Electronics* **2024**, *13*, 1004.
11. Gollapalli, V.S.T.; Srinivasan, K.; Chauhan, G.S.; et al. Cybersecurity Attack Detection Using LSTM and ResNet-50 Hybrid Model with Cloud Deployment. *Indo-Am. J. Mech. Eng.* **2023**, *12*, 33–46.
12. Hernández, D.; Cecilia, J.M.; Cano, J.C.; et al. Flood Detection Using Real-Time Image Segmentation from Unmanned Aerial Vehicles on Edge-Computing Platform. *Remote Sens.* **2022**, *14*, 223.
13. Rashid, K.; Saeed, Y.; Ali, A.; et al. An Adaptive Real-Time Malicious Node Detection Framework Using Machine Learning in Vehicular Ad-Hoc Networks (VANETs). *Sensors* **2023**, *23*, 2594.
14. Yallamelli, A.R.G.; Mamidala, V.; Devarajan, M.V.; et al. Dynamic mathematical hybridized modeling algorithm for e-commerce for order patching issue in the warehouse. *Serv. Oriented Comput. Appl.* **2024**, 1–12. https://doi.org/10.1007/s11761-024-00431-w
15. Villegas-Ch, W.; Jaramillo-Alcázar, A.; Luján-Mora, S. Evaluating the Robustness of Deep Learning Models against Adversarial Attacks: An Analysis with FGSM, PGD and CW. *Big Data Cogn. Comput.* **2024**, *8*, 8.
16. Dawood, M.; Tu, S.; Xiao, C.; et al. Cyberattacks and Security of Cloud Computing: A Complete Guideline. *Symmetry* **2023**, *15*, 1981.
17. Sitaraman, S.R.; Khalid, H.M. Robotics automation and adaptive motion planning: A hybrid approach using AutoNav, LIDAR-based SLAM, and DenseNet with Leaky ReLU. *J. Trends Comput. Sci. Smart Technol.* **2024**, *6*, 404–423.
18. Awajan, A. A Novel Deep Learning-Based Intrusion Detection System for IoT Networks. *Computers* **2023**, *12*, 34.
19. Umer, M.; et al. Deep Learning-Based Intrusion Detection Methods in Cyber-Physical Systems: Challenges and Future Trends. *Electronics* **2022**, *11*, 3326.

20. Basani, D.K.R.; Grandhi, S.H.; Abbas, Q. Centralized infrastructure-aware reliable data transaction model in IoT-enabled MANET and cloud using GWO and attention mechanism with LSTM. *Int. J. Adv. Res. Inf. Technol. Manag.* **2024**, *1*, 110–134.
21. Qazi, E.U.H.; Faheem, M.H.; Zia, T. HDLNIDS: Hybrid Deep-Learning-Based Network Intrusion Detection System. *Appl. Sci.* **2023**, *13*, 4921.
22. Alzubi, O.A.; Alzubi, J.A.; Alazab, M.; et al. Optimized Machine Learning-Based Intrusion Detection System for Fog and Edge Computing Environment. *Electronics* **2022**, *11*, 3007.
23. Kavitha, C.; S. M.; Gadekallu, T.R.; et al. Filter-Based Ensemble Feature Selection and Deep Learning Model for Intrusion Detection in Cloud Computing. *Electronics* **2023**, *12*, 556.
24. R M, B.; M K, J.K. Intrusion Detection on AWS Cloud through Hybrid Deep Learning Algorithm. *Electronics* **2023**, *12*, 1423.
25. Parthasarathy, K. Enhanced case-based reasoning with hybrid clustering and evolutionary algorithms for multi-class workload forecasting in autonomic database systems. *Int. J. HRM Organ. Behav.* **2023**, *11*, 38–54.
26. Ramachandran, D.; Albathan, M.; Hussain, A.; et al. Enhancing Cloud-Based Security: A Novel Approach for Efficient Cyber-Threat Detection Using GSCSO-IHNN Model. *Systems* **2023**, *11*, 518.
27. Javed, A.; Ehtsham, A.; Jawad, M.; et al. Implementation of Lightweight Machine Learning-Based Intrusion Detection System on IoT Devices of Smart Homes. *Future Internet* **2024**, *16*, 200.
28. Alduailij, M.; Khan, Q.W.; Tahir, M.; et al. Machine-Learning-Based DDoS Attack Detection Using Mutual Information and Random Forest Feature Importance Method. *Symmetry* **2022**, *14*, 1095.
29. ElKashlan, M.; Elsayed, M.S.; Jurcut, A.D.; et al. A Machine Learning-Based Intrusion Detection System for IoT Electric Vehicle Charging Stations (EVCSs). *Electronics* **2023**, *12*, 1044.
30. Rangelov, D.; et al. Towards an Integrated Methodology and Toolchain for Machine Learning-Based Intrusion Detection in Urban IoT Networks and Platforms. *Future Internet* **2023**, *15*, 98.
31. Induru, V.; Arulkumaran, G. Adaptive cybersecurity monitoring via semantic stream processing and GNN-based trust scoring on IPv4 logs. *Int. J. Bus. Manag. Econ. Rev.* **2021**, *4*, 430–443.
32. Deshmukh, A.; Ravulakollu, K. An Efficient CNN-Based Intrusion Detection System for IoT: Use Case Towards Cybersecurity. *Technologies* **2024**, *12*, 203.
33. Zhong, W.; Yu, N.; Ai, C. Applying big data based deep learning system to intrusion detection. *Big Data Min. Anal.* **2020**, *3*, 181–195.
34. Valivarthi, D.T.; Peddi, S.; Narla, S.; et al. Security-aware side-channel detection through convolutional transformer networks and hybrid LSTM-spectral analysis. *Int. J. Adv. Res. Inf. Technol. Manag. Sci.* **2024**, *1*, 17–24.
35. Adel, A.; Jan, T. Watch the Skies: A Study on Drone Attack Vectors, Forensic Approaches, and Persisting Security Challenges. *Future Internet* **2024**, *16*, 250.
36. El-Gayar, M.M.; Alrslani, F.A.F.; El-Sappagh, S. Smart Collaborative Intrusion Detection System for Securing Vehicular Networks Using Ensemble Machine Learning Model. *Information* **2024**, *15*, 583.
37. Nippatla, R.; Vasamsetty, C.; Kadiyala, B.; et al. Next-generation healthcare frameworks: Lightweight CNNs, capsule networks, and blockchain alternatives for real-time pandemic detection and data security. *J. Ubiquitous Comput. Commun. Technol.* **2024**, *6*, 407–428.
38. Vasani, V.; Bairwa, A.K.; Joshi, S.; et al. Comprehensive Analysis of Advanced Techniques and Vital Tools for Detecting Malware Intrusion. *Electronics* **2023**, *12*, 4299.
39. Srinivasan, K.; Chauhan, G.S.; Jadon, R.; et al. A Real-Time AI-Driven Surgical Monitoring Platform Using Robotics, 3D Convolutional Neural Networks (3D-CNNs), and Bayesian Optimization for Enhanced Precision. In Proceedings of the 2024 International Conference on Computing and Intelligent Reality Technologies (ICCIRT), Coimbatore, India, 5–6 December 2024; pp. 1–6.
40. Awad, M.; Fraihat, S. Recursive Feature Elimination with Cross-Validation with Decision Tree: Feature Selection Method for Machine Learning-Based Intrusion Detection Systems. *J. Sens. Actuator Netw.* **2023**, *12*, 67.
41. Aldallal, A. Toward Efficient Intrusion Detection System Using Hybrid Deep Learning Approach. *Symmetry* **2022**, *14*, 1916.
42. Dyavani, N.R.; Ubagaram, C.; Garikipati, V.; et al. Adaptive access control in SHACS: Leveraging Markov models and topological data analysis for enhanced cloud security. *Int. J. Innov. Technol. Creat. Eng.* **2024**, *12*, 205–225.
43. Meliboev, A.; Alikhanov, J.; Kim, W. Performance Evaluation of Deep Learning Based Network Intrusion Detection System across Multiple Balanced and Imbalanced Datasets. *Electronics* **2022**, *11*, 515.
44. Shanmugam, V.; Razavi-Far, R.; Hallaji, E. Addressing Class Imbalance in Intrusion Detection: A Comprehensive Evaluation of Machine Learning Approaches. *Electronics* **2024**, *14*, 69.
45. Ramar, V.A.; Kushala, K.; Induru, V.; et al. AI-Augmented Test Automation: Integrating Page Object Model and Behavior-Driven Development for Intelligent and Scalable Software Testing. *Int. J. Multidiscip. Res. Growth Eval.* **2020**, *5*, 1078–1085.

46. Ahmad, I.; Basheri, M.; Iqbal, M.J.; et al. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE Access* **2018**, *6*, 33789–33795.
47. Alrayes, F.S.; Zakariah, M.; Alzaylaee, M.K.; et al. Optimizing Intrusion Detection System (IDS) with Hybrid Random Forest and CNN-LSTM Models for Improved Accuracy and Efficiency. 2025. Available online: https://www.researchsquare.com/article/rs-6766340/v1 (accessed on).
48. Adewole, K.S.; Jacobsson, A.; Davidsson, P. Intrusion detection framework for Internet of Things with rule induction for model explanation. *Sensors* **2025**, *25*, 1845. https://doi.org/10.3390/s25061845.
49. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 1153–1176.
50. Ali, M.L.; Thakur, K.; Schmeelk, S.; et al. Deep learning vs. machine learning for intrusion detection in computer networks: A comparative study. *Appl. Sci.* **2025**, *15*, 1903.
51. Fan, Z.; You, Z. Research on network intrusion detection based on XGBoost algorithm and multiple machine learning algorithms. *Theor. Nat. Sci.* **2024**, *31*, 161–166.